REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
	Professional Paper	

4. TITLE AND SUBTITLE

Integration of the CASTLE Simulation Executive with Simulink

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Thomas Magyar
Anthony Page

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Air Warfare Center Aircraft Division
22347 Cedar Point Road, Unit #6
Patuxent River, Maryland 20670-1161

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Controls Analysis and Simulation Test Loop Environment (CASTLE) is a Navy in-house developed aircraft simulation executive application that has been in use at the Manned Flight Simulator (MFS) facility in Patuxent River, Maryland and elsewhere for over 15 years. In addition to supporting real-time hi-fidelity pilot-in-the-loop simulations in the MFS facility, CASTLE also includes integrated tools used for development and analysis of airframe simulations on various platforms.

Much of the work with CASTLE includes exchanging and analyzing data with Matlab and Simulink. In support of recent flight controls projects at Patuxent River, there was a requirement to integrate Matlab-Simulink models with the CASTLE simulation. This led to the development of a new capability that enables a Simulink model to be modified and executed continuously with the CASTLE simulation, without having to exit and restart. The only interaction with CASTLE required by the user is to launch the specific airframe executable desired.

15. SUBJECT TERMS

Controls Analysis and Simulation Test Loop Environment (CASTLE); Simulink; Matlab

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Thomas Magyar / Anthony Page
				8	19b. TELEPHONE NUMBER (include area code) (301) 757-0852 / 342-8566

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

AIAA-2001-4121

INTEGRATION OF THE CASTLE SIMULATION EXECUTIVE WITH SIMULINK

Thomas J. Magyar*
Anthony B. Page†
Naval Air Systems Command, Patuxent River, MD

Abstract

The Controls Analysis and Simulation Test Loop Environment (CASTLE) is a Navy in-house developed aircraft simulation executive application that has been in use at the Manned Flight Simulator (MFS) facility in Patuxent River, Maryland and elsewhere for over 15 years. In addition to supporting real-time hi-fidelity pilot-in-the-loop simulations in the MFS facility, CASTLE also includes integrated tools used for development and analysis of airframe simulations on various platforms.

Much of the work with CASTLE includes exchanging and analyzing data with Matlab and Simulink. In support of recent flight controls projects at Patuxent River, there was a requirement to integrate Matlab-Simulink models with the CASTLE simulation. This led to the development of a new capability that enables a Simulink model to be modified and executed continuously with the CASTLE simulation, without having to exit and restart. The only interaction with CASTLE required by the user is to launch the specific airframe executable desired.

Using an S-Function (System function) user definable block in Simulink, a Matlab C-MEX file was created to interface Simulink with the CASTLE airframe using TCP/IP networking. In this scheme, when the Simulink model is executed and the S-Function block is called, a connection to CASTLE is established and a list of input and output variable names and values are sent for setup. The input values to the S-Function are then sent across the connection to the CASTLE airframe simulation. The simulation is then run through a simulation frame, halted, and output values requested are sent across the connection as the outputs to the S-Function block. This loop is continued for the duration of the Simulink model end run time, at which point the connection is terminated. In this manner, a hi-fidelity airframe simulation running under CASTLE can be used to develop, test and evaluate Simulink models of any type.

Background

The design of the CASTLE software is based on a C/C++ graphical user interface (GUI) and a C and FORTRAN airframe executive shell. The CASTLE environment was designed to be platform independent and is currently hosted on PC's running Microsoft Windows 98/NT/2000 operating system, DEC Alpha's running the VMS operating system, and SGI's running the Unix operating system.

The design of the CASTLE software is very modular on several levels. It consists of several tasks running simultaneously, with the CASTLE GUI being the controlling entity of the simulation in most instances. Other processes include the actual airframe simulation, the plotting package and a 3D visualization software package. The airframe is used for analysis and development by simulation engineers and in real time pilot-in-the-loop simulations. These processes communicate via several methods, including local or global shared memory, and a TCP/IP protocol. In addition to these specific processes, CASTLE can communicate to other external processes by using either of these methods.

The TCP/IP protocol used with CASTLE is the Data Transfer Mechanism (DTM) designed by the University of Illinois National Center for Supercomputing Applications (NCSA). The use of this protocol allows the CASTLE tasks to be running on separate machines if required. Data word format conversions of the information packets transferred between these tasks are performed automatically. This was the method selected to best communicate with the Matlab/Simulink process. Figure 1 shows the CASTLE processes and the communication methods among them.

* Aerospace Engineer, Member AIAA
† Aerospace Engineer, Member AIAA

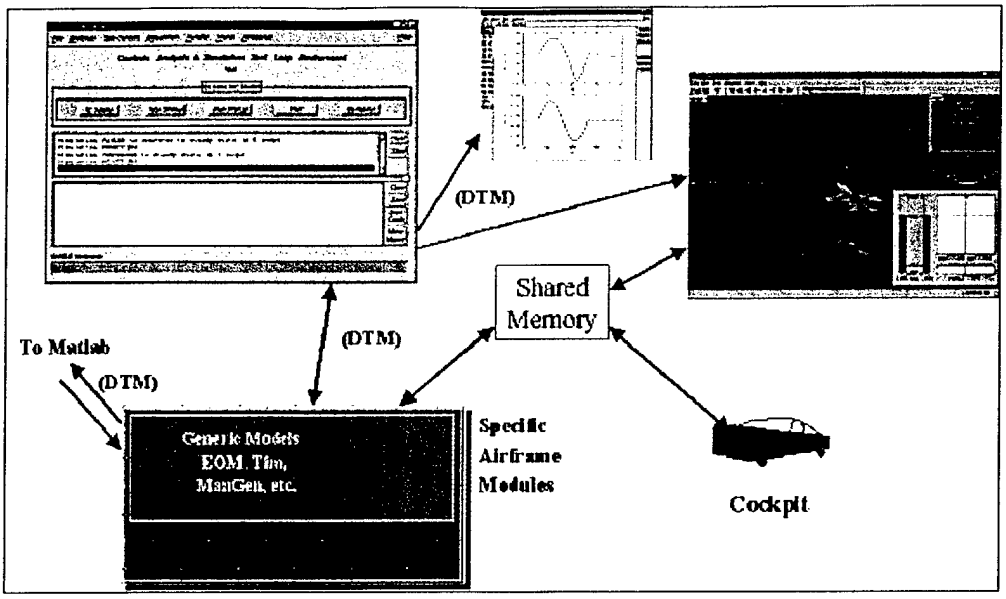

Figure 1: CASTLE Design and Communication

Purpose

Much of the work with CASTLE includes exchanging and analyzing data with Matlab and Simulink. CASTLE has the ability to import and export data in the Matlab M-file or MAT-file formats. To expand upon this current capability and in support of a recent flight controls project, it was proposed to establish a Matlab interface with CASTLE, with priority being to communicate with a working Simulink model.

It was desired to use the hi-fidelity CASTLE airframe simulation in place of a lower fidelity Simulink model of the airframe. In this configuration, when executing the Simulink model, the CASTLE airframe simulation would obtain inputs and supply outputs to the rest of the Simulink model. This would continue for the duration of the model's execution.

During development, the Simulink model can undergo many changes. In light of this, it was desired to avoid terminating the current simulation session and recompiling the airframe source code with each change. This would greatly facilitate rapid development and testing.

Most users of this new tool will be experienced with Matlab/Simulink and will be doing most of their development and analysis in this environment. It was desired to control the CASTLE simulation from within the Matlab/Simulink application, thus minimizing the time spent switching between applications.

Options

During the initial design phase, two options were considered that could possibly fulfill the requirements.

Matlab ActiveX Engine

This method uses the Matlab API programmed from within CASTLE to send commands to the Matlab workspace. In this scenario, Matlab is treated as the "server" and CASTLE as the "client". In this mode, all model setup and execution would be done from within CASTLE, with no direct interaction with Matlab necessary.

From within CASTLE, a new facility (GUI window) would be created to setup the variables to transfer, the Matlab commands to execute during the simulation run, and other options to set before execution. Commands could also specify which Simulink model to load at simulation run time. During execution of the normal simulation loop, CASTLE would divert to the Simulink model at the appropriate time, sending the necessary variable values to the workspace. The execution commands would then be sent, running the Simulink model. After one iteration of the Simulink model, CASTLE would request variable values from the workspace and set them to the appropriate simulation variables. The loop would continue until simulation execution is complete.

The Matlab API is straightforward to implement in this fashion. However, it was unclear how to run a Simulink model for one iteration and preserve the new state

information, for use during the next loop. Also, this option would require the user to perform all setup and model execution from within CASTLE, requiring the user to toggle back and forth among applications when developing the Simulink model.

Matlab C-Mex Files and DTM

This method uses C-Mex files together with a Simulink S-function block and CASTLE's DTM to transfer information between the processes. In this mode all model setup and execution of the CASTLE simulation can be accomplished from within the Matlab workspace with no interaction with the CASTLE GUI required.

Placing this CASTLE S-function block in a Simulink model creates a connection to the CASTLE airframe when executed. Variables to send and receive from the airframe are declared in the Matlab workspace.

When the Simulink model is executed, a TCP/IP connection is established to the airframe. The variables declared in the Matlab workspace are sent to CASTLE for initialization and a flag is set indicating that a Simulink model is now part of the CASTLE airframe simulation loop. Input values to the S-function are passed to the airframe to overdrive the simulation and after an iteration of the airframe loop, variable values are passed back as outputs to the S-function. This process continues until the Simulink model reaches its stop time.

Using this method, the control of the simulation could be maintained in Matlab. Since the Simulink model could then be run from start time to stop time, the calculations and state information could be preserved through each iteration. This was chosen as the preferred method to fulfill the requirements.

S-Function Design

Under normal circumstances, CASTLE creates a set of TCP/IP ports to communicate with the airframe, and another set for use with the 3D visualization software, CasView. To communicate with Matlab another set of ports are created on the airframe at program launch, as shown in Figure 1. The ports are periodically checked for a Matlab request for connection. Once CASTLE has been launched, no further interaction with it is necessary.

S-Function Block Setup

In Simulink, an S-function is a general block that can be customized to suit the needs that any other Simulink block can't provide. The S-function block is associated with a source file that is programmed to define it's behavior. In this instance, the source file was written in C and compiled as a MEX-file. This creates a dynamically linked library (DLL) that is called when the Simulink model is executed.

To enable the CASTLE airframe to be included in the Simulink model, a custom S-function block was created. This block contains three extra input parameters, in addition to the standard t, x and u that is supplied with any Simulink S-function.

The first parameter is a cell array of variable names that are used to indicate to the airframe what variable values are being passed as inputs before each simulation loop. This array can be of any size.

The second parameter is a cell array of variable names that are used to indicated what variables are to be returned as outputs after the simulation loop. This array also can be of any size.

The third parameter is a cell array listing any miscellaneous values that are to be set in the airframe model before the run is initiated. The values are retrieved from variables with the same name in the Matlab workspace during S-function initialization and sent to the airframe for deposit into the simulation. This array also can be of any size.

The CASTLE S-function block contains one input port and one output port, as is always the case with S-function blocks. To accommodate the arbitrary width arrays of the input and output parameters, the S-function was coded to handle dynamically sized inputs and outputs. However, with any dynamically sized S-function, the input vector width is used to determine the output vector width, so these numbers must be identical.

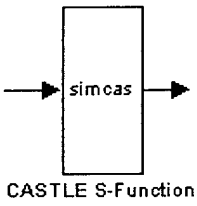

Figure 2: CASTLE S-function block

To resolve this issue and still allow an arbitrary number of inputs and outputs, the S-function was written to accept, and ignore, zero array entries. This "padding" can be placed on either the input or output vectors to obtain the same overal vector width.

Model Execution

With the CASTLE S-function in the Simulink model, the input and output parameters specified, the miscellaneous values declared and set in the workspace, and the CASTLE airframe simulation launched, execution of the model can begin.

Initialization

When the Simulink model is executed, a connection is attempted with the CASTLE airframe using the DTM library. If CASTLE is found and is available, the connection is established.

If successful, the input parameter variable name is retrieved from Matlab workspace. This cell array is then parsed and a list of variable names used for input is sent to the airframe. The airframe checks to be sure that these variables exist in the simulation before continuing. These values are then stored in a list by the airframe. This same method is used to set and confirm the output variables.

If a third parameter contains entries, these variable names will be sent to the airframe just as the previous input and output variables. In addition, the S-function will look up each variable name in the workspace, obtain the values, and send them to the airframe. The airframe will set the values retrieved to the simulation variables before start of the model.

Execution of the Simulink Model

With initialization complete, the Simulink model will begin executing its normal loop, calling the CASTLE S-function. A pointer to the input and output array values are passed into this function.

If a connection has been established and initialization is complete the array of input value's to the S-function are sent to the CASTLE airframe using DTM. The S-function will then wait for a response from the airframe, indicating it has completed the simulation loop iteration and is passing back the required output values.

As the Simulink model is waiting for the outputs, the airframe obtains the inputs from the S-function. This array of inputs must match the order that the variable names were given during initialization. The inputs are then added to the overdrive list and used to overdrive the simulation with the current values through the simulation loop.

A mode flag variable is used in CASTLE to determine if a Simulink model is being used, which modifies the run loop appropriately, and is set at this time. With the flag set, the simulation will run through one iteration and then set the halt command, notifying the airframe to halt its run loop at the end of the current iteration. By halting, or pausing the simulation all variable value data is preserved for the next iteration.

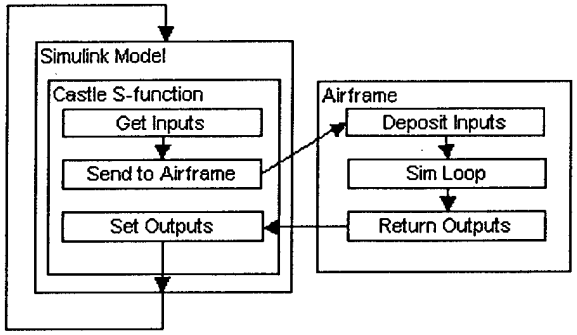

Figure 3: Model Execution

After the loop is complete, the values from the output variables requested during initialization are retrieved and sent back to the CASTLE S-function in the Simulink model. The CASTLE airframe will then wait for the next packet of information from the Simulink model. The next packet will either be another input values packet, indicating another iteration of the simulation loop, or a termination packet, indicating the run is complete.

Control then transfers back to the S-function. Upon receiving the output values packet, the values are unpackaged and set as the output values from the S-function. The Simulink model then continues, repeating this order of execution until the end time is reached. Figure 3 illustrates the complete process.

Upon termination a packet is sent to the CASTLE airframe indicating completion of the run. The airframe resets the mode flag and both terminate the connection to one another.

A Utility Function

In order to set the initial conditions and trim the airframe before a run, a separate utility function was developed as a C-Mex file, and is used to set and issue commands to the CASTLE airframe from within Matlab. This can be executed by the user before running the Simulink model. This function has three arguments.

The first argument contains the command to perform. Possible commands currently are "Trim" and "Reset".

The second argument is a cell array of the airframe state variable names whose values are to be set. The values are retrieved from variables with the same name in the Matlab workspace.

The third argument is an array of variable names whose values should be passed back to the Matlab workspace after the command has been executed. This argument is optional. If specified, these values will be returned in an array to the left-hand side of the equation. The values in the returned array are in the same order as those in the third argument. These values can then be used to set variable values in the Matlab workspace or Simulink model. Figure 4 illustrates the method of calling the function, either from the Matlab command prompt, or an M-file.

```
results = execute_cas( command, in, out );
```

Figure 4: Calling 'execute_cas()' function

Example Applications

The S-function shown in Figure 2 allows the use of the highest fidelity CASTLE airframe models directly in Matlab/Simulink simulations. This new capability should prove beneficial for a wide variety of users. For users familiar with Matlab/Simulink but not with CASTLE, the new capability allows access to the highest fidelity airframe models with very little spool up time. For the control law and flying qualities analysts who do much of their analyses in Matlab, this will lead to less switching back and forth between applications when running simulations interactively. For control law researchers and developers, this will allow the design and modification of control laws to be carried out in Matlab/Simulink and then tested directly using the highest fidelity airframe model available. Therefore, control law researchers and developers can take advantage of Matlab's powerful features and Simulink's graphical interface when developing control laws and then rapidly transition to testing the designs using the CASTLE simulation model without re-coding the control law designs in C or FORTRAN.

Since the CASTLE S-function can be incorporated into any Simulink simulation, it should be useful in a variety of engineering design and analysis problems. However, a couple of examples should sufficiently illustrate some of the potential applications.

Pilot Input

Aircraft flight control systems and flying qualities are often evaluated based on vehicle response to pilot inputs.[4] These inputs can be open-loop such as a step, doublet, or sine sweep or the inputs can be the result of performing some closed-loop maneuver such as a bank angle capture or a windup turn to a specified load factor.

As a simple example, consider a longitudinal stick doublet as illustrated in the Simulink diagram shown in Figure 5. In order to run this Simulink simulation, the CASTLE airframe simulation must first be launched. After the desired CASTLE model is launched, no further direct interaction with CASTLE is required. However, the user can interact with CASTLE as desired to set the aircraft loading and initial conditions. For the current example, a settings file is loaded from within CASTLE that places a clean F/A-18C at a low dynamic pressure flight condition. At this point, the CASTLE model may either be trimmed interactively or from within Matlab using the 'execute_cas' function of Figure 4.

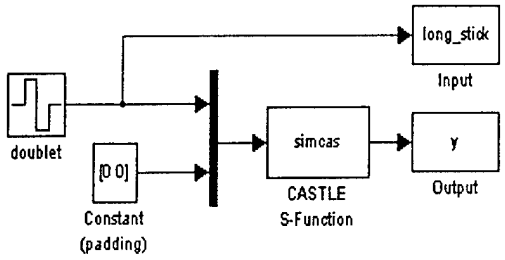

Figure 5: Simulink Diagram for Pilot Input

Now, all that is required within Matlab before running the Simulink simulation is to set the inputs, outputs, and miscellaneous variables used by the CASTLE S-function. For the F/A-18C/D, the following Matlab commands are issued:

cas_inputs = {'STLON'}
cas_outputs = {'QB', 'THET', 'ALFA'}
cas_vars = {'DO_STORE'}
DO_STORE = 1

The first command sets the input to be the longitudinal stick position. The second command sets the outputs to be the pitch rate, pitch angle, and angle of attack. The next command tells the CASTLE S-function to get the value for the CASTLE variable DO_STORE from the Matlab workspace. The final command sets this value to 1 in the Matlab workspace and is used to instruct CASTLE to save its digital output. Since the CASTLE

S-function requires an equal number of inputs and outputs, the input to the CASTLE S-function must be padded with a two-element vector whose values will be ignored by the model (See Figure 5).

Now, the Simulink simulation can be run in the usual manner. Note, with the current formulation of the CASTLE S-function, the Simulink model is required to use the first order ordinary differential equation solver with a fixed step size equal to that specified in the CASTLE simulation. Future versions of the CASTLE S-function should relax this requirement.

The results from running the simulation are given in Figure 6 and show the normalized pitch rate response due to the longitudinal stick doublet. As a point of interest, if it were also desired to output the load factor response, then the *cas_outputs* cell array would simply be appended with the appropriate variable name. Since this would bring the total number of outputs to four, the input padding vector would also need to be increased to contain a total of three elements. The Simulink simulation could then be re-run without any need for the user to directly interact with CASTLE. This ability to modify the input/output of the CASTLE S-function as well as to modify the Matlab/Simulink model on the fly (i.e., without having to stop or re-start either Matlab or CASTLE) should prove exceptionally valuable to the user.

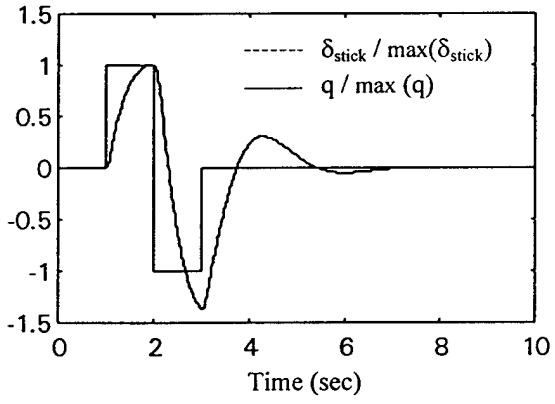

Figure 6: Pitch Rate Response to Stick Doublet

Adaptive Control

In this example it is desired to analyze the performance of a nonlinear adaptive control approach used in conjunction with an advanced control allocation routine.[5] The top level Simulink diagram is shown in Figure 7. Here, a third order command generator is used to provide smoothly varying values of bank angle, angle of attack, and angle of sideslip that the controller is designed to track.

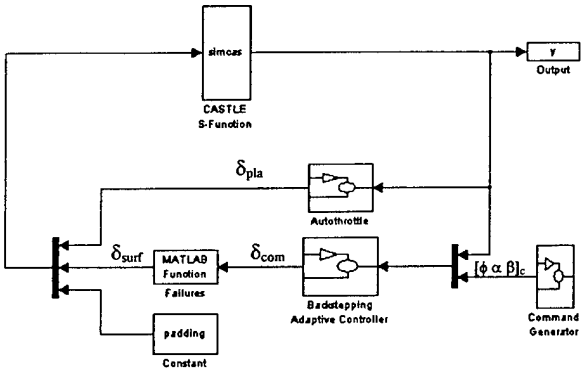

Figure 7: Simulink Diagram for Adaptive Control

The control law is an adaptive backstepping design based on a two-loop dynamic inversion controller. In the outer loop, desired values of the angular rates ($\omega = [p\ q\ r]^T$) are used as virtual controls to perform tracking of the outputs (ϕ, α, β). In the inner loop, tracking of the desired angular rates leads to the control allocation problem that must be solved to determine the control effector commands (δ_{com}). Various stability derivatives and control effectiveness parameters are estimated on-line using a Lyapunov approach to ensure convergence of the tracking error. A failure module is used to inject actuator failures in the command path and a separate autothrottle is used to provide throttle commands (δ_{pla}) in an attempt to maintain constant velocity.

In this case, the inputs to the CASTLE S-function consist of the throttle and surface actuator commands. The output of the CASTLE S-function consists of the normal aircraft state information as well as other values of interest. Again, the input is padded as needed to give an equal number of inputs and outputs. In contrast to the example with pilot input, the production control law calculations in the CASTLE simulation are bypassed as the CASTLE simulation is being overdriven by the actuator commands calculated by the Simulink model's controller. If desired, CASTLE's actuator model calculations can also be bypassed by selecting the surface positions instead of the actuator commands as inputs to the CASTLE S-function.

Again, before the Simulink model simulation can be run, the CASTLE airframe model must be launched and any desired settings file loaded. Since the adaptive controller needs information about the initial state of the airframe model, the 'execute_cas' function is used to trim the simulation. Namely,

results = execute_cas('TRIM', *in_vars*, *out_vars*)

where *in_vars* is a cell array containing the variable names for altitude and airspeed and *out_vars* is a cell array containing the variable names for the throttle position, surface positions, and a flag that indicates whether or not the trim operation is successful. Before the 'execute_cas' command is issued, the desired altitude and airspeed are defined in the Matlab workspace using the associated CASTLE variable names. After the command is executed, the trim positions are read from the *results* vector and passed to the adaptive controller for initialization.

After the CASTLE model has been trimmed as described above and the input and output variable names for the CASTLE S-function have been defined in the Matlab workspace, the Simulink simulation can be executed as normal. Figure 8 shows the tracking results for a low dynamic pressure flight condition using the adaptive backstepping control law with a Direct Allocation control allocator.[5] Tracking performance is seen to be very good with all outputs closely tracking the commanded values. Figure 9 shows results for the same commanded maneuver but with an aileron failure (hardover) one second into the simulation. Although there is some performance degradation, the controller remains stable and provides good tracking response.

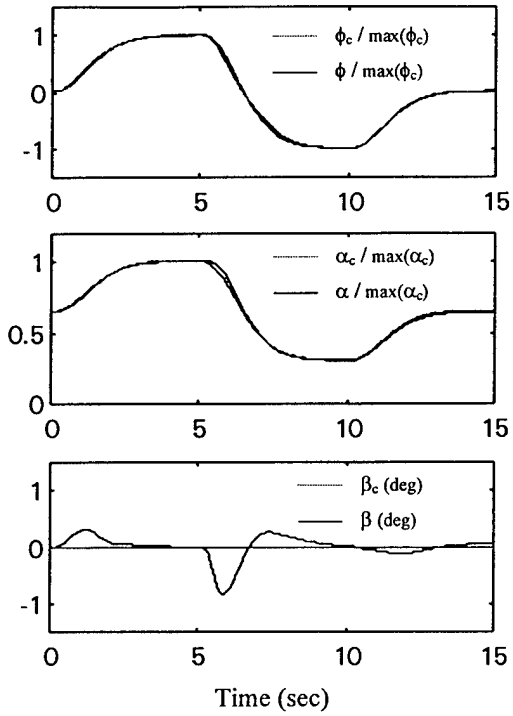

Figure 8: Tracking Results (No Failures)

Conclusions

Combining the many capabilities of the CASTLE simulation environment with Matlab and Simulink was successful and accomplished in a manner to allow the use of any Simulink model with any CASTLE airframe model. The high-fidelity CASTLE airframe model can now operate as part of the overall Simulink model, being controlled from within Matlab. This integrates all setup and development to the Matlab application by using the CASTLE S-function and the 'execute_cas' utility function. Once CASTLE and the airframe are initially launched, the Simulink model can be modified continuously and re-run without having to perform a rebuild or relink of the airframe source code. The airframe and Matlab/Simulink operate independently until the time of model execution allowing the option of using CASTLE or Simulink in its normal standalone fashion if desired between integrated runs. This new capability to CASTLE versions 5.6 and later should greatly enhance the ability to develop, test and evaluate Simulink models with a hi-fidelity airframe model.

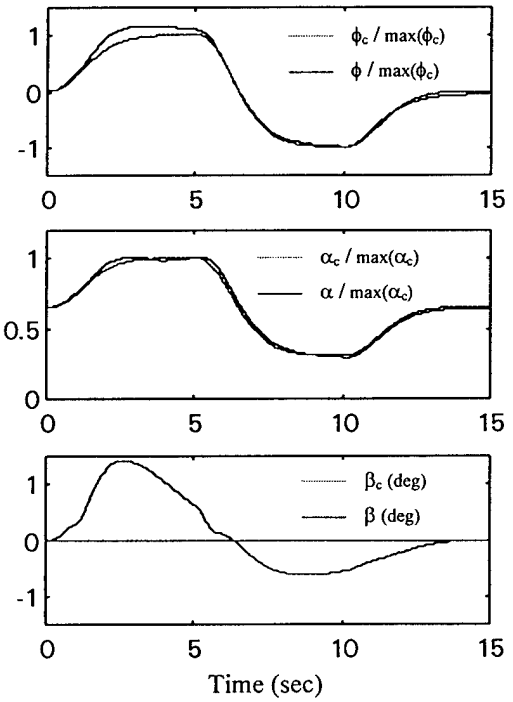

Figure 9: Tracking Results (Aileron Hardover)

References

[1] Nichols, J.H., Magyar, T.J., Schug, E.C., "The Platform Independent Aircraft Simulation Environment at Manned Flight Simulator", AIAA Paper No. 98-4179, AIAA Modeling and Simulation Technologies Technology Conference, Boston, MA, 1998.

[2] Mathworks Inc., "Using Simulink", Simulink 2 Documentation, 1997

[3] Mathworks Inc., "Application Program Interface Guide", Matlab Documentation, 1998.

[4] MIL-STD-1797A, Military Standard, Flying Qualities of Piloted Aircraft, 30 January, 1990.

[5] Page, A., and Steinberg, M., "Effects of Control Allocation Algorithms on a Nonlinear Adaptive Design," AIAA Paper No. 99-4282, AIAA Guidance, Navigation, and Control Conference, Portland, OR, 1999.